Diffusing ideas

Adventures with noise and building mathematical toys

Robert J. Hardwick

January 4, 2023

Introduction

Diffusing ideas records a journey of research exploration and software development. It expresses a collection of interrelated ideas around the statistical inference and automated control of stochastic phenomena. In order to manifest these ideas into reality, this project has also involved designing and building a lot of new open-source scientific software written in the Python and Go progamming languages. My primary motivation behind developing these computational elements is to form a nice foundation from which anyone can build loads of new applications, and I hope that an additional benefit should be that they make scientific research and discovery of new phenomena a much more efficient experience for the scientist.

Having just stated all this 'altruistic' stuff, I would be remiss if I left the motivations for writing this book at that. The necessity for testing the software has provided a great excuse to study, play with and invent new derivatives of an extensive range of toy mathematical models and imagining some of these situations where the code might be useful has been a lot of fun. I'm also not going to write an 'about the author' section because I think they are usually quite boring. Suffice it to say that I have a background in various fields of science which I hope will make the mathematical/computational digressions quite varied and interesting to the reader!

A quick note on the software itself. Any software that I describe in this book (including the software which compiles the book itself) will always be shared under a MIT License in the following public git repository: https://github.com/umbralcalc. Forking these repositories and submitting pull requests for new features or applications is strongly encouraged too, though I apologise in advance if I don't follow these up very quickly as all of this work has been conducted independently in my free time outside of work hours.

No quest would be complete without a map, so in this introductory section, I think it makes the most sense to now outline some of the key milestones of the book and their motivations within the context of the overall research project. My core aims can be divided into proposing answers to a set of interdependent research questions, in the following order:

1. How do we simulate a general set of stochastic phenomena?

2. How do we then learn/identify 1. from real-world data?

3. How do we simulate a general set of automated control policies to interact with 1.?

4. How do we then optimise 3. to achieve a specified control objective?

Table of contents

1

Building the ‘stochadex’

Concept. We will design and build a generalised simulation engine that is able to generate samples from a ‘Pokédex’ of possible stochastic processes that a researcher might encounter. A ‘Pokédex’ here is just a fanciful description for a very general class of multidimensional stochastic processes that pop up everywhere in taming the mathematical wilds of real-world phenomena, and which also leads to a name for the software: the stochadex. With such a thing pre-built and self-contained, it can become the basis upon which to build generalised software solutions for a lot of different problems.

1.1 Core mathematical formalism

Ideally, the stochadex sampler should be designed to try and maintain a balance between performance and flexibility of utilisation. But before we dive into the software, we need to define the mathematical approach we're going to take. It seems reasonable to start with writing down the following formula which describes iterating some arbitrary process forward in time (by one finite step) and adding a new row each to some matrices $V' \rightarrow V$ and $X' \rightarrow X$

$$X^i_{t+1} = F^i_{t+1}(X', V', t) \tag{1.1}$$

$$V^i_{t+1} = X^i_{t+1} - X^i_t \,, \tag{1.2}$$

where: i is an index for the dimensions of the ‘state’ space; t is the current time index for either a discrete-time process or some discrete approximation to a continuous-time process; X is the next version of X' after one timestep (and hence one new row has been added); V as the next version of V' (similarly to X and X'); and $F^i_{t+1}(X', V', t)$ as the latest element of an arbitrary matrix-valued function.

So the idea here is to iterate the matrices X and V forward in time by a row, and use their previous versions (X' and V') as entire matrix inputs into functions which populate the elements of their latest rows.

Note that, based on the definition in Eq. (1.2) above, the following relation is also valid

$$X^i_t = X^i_s + \sum_{s'=s+1}^{t} V^i_{s'} \,, \tag{1.3}$$

where $s < t$.

Why go to all this trouble of storing matrix inputs for previous values of the same process? For a large class of stochastic processes this memory of past values is essential to consistently construct the sample paths moving forward. This is true in particular for *non-Markovian* phenomena (see, e.g., Markov chains to get a sense of their antithesis), where the latest values don't just depend on the immediately previous ones but can depend on values which occured much earlier in the process.

1.2 Flavours of noise with continuous sample paths

For *Wiener process noise*, adopting the Itô interpretation in this section, we can define W^i_t is a sample from a Wiener process for each of the state dimensions indexed by i and our formalism becomes

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + W^i_{t+\delta t} - W^i_t \,. \tag{1.4}$$

Other interpretations of the noise are less immediately compatible with our formalism as it is currently written, e.g., Stratonovich or others within the α-family, but it seems less necessary to complicate the details of this section further, so we'll just cover these extensions at the software implementation level. Note also that we may also allow for correlations between the noises in different dimensions.

For *Geometric Brownian motion noise*, we simply have

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + X^i_t(W^i_{t+\delta t} - W^i_t) \,. \tag{1.5}$$

And say, e.g., *fractional Brownian motion noise*, where $B^i_t(H_i)$ is a sample from a fractional Brownian motion process with Hurst exponent H_i for each of the state dimensions indexed by i, we simply substitute again

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + B^i_{t+\delta t}(H_i) - B^i_t(H_i) \,. \tag{1.6}$$

Generalised continuous noises would take the form

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + g^i_{t+\delta t}(X', W^i_{t+\delta t} - W^i_t, \dots) \,, \tag{1.7}$$

where $g^i_{t+\delta t}(X', W^i_{t+\delta t} - W^i_t, \dots)$ is some continuous function of its arguments which can be expanded out with Itôs Lemma.

1.3 Flavours of noise with discontinuous sample paths

Jump process noises generally could take the form

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + J^i_{t+\delta t}(X', \dots) \,, \tag{1.8}$$

where $J^i_{t+\delta t}(X', \dots)$ are samples from some arbitrary jump process (e.g., compound Poisson) which could generally depend on a variety of inputs, including X'.

Poisson process noises would generally take the form

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + N^i_{t+\delta t}(\lambda_i) - N^i_t(\lambda_i)\,, \tag{1.9}$$

where $N^i_t(\lambda_i)$ is a sample from a Poisson process with rate λ_i for each of the state dimensions indexed by i. Note that we may also allow for correlations between the noises in different dimensions.

Time-inhomogeneous Poisson process noises would generally take the form

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + N^i_{t+\delta t}(\lambda^i_{t+\delta t}) - N^i_t(\lambda^i_t)\,, \tag{1.10}$$

where λ^i_t is a deterministically-varying rate for each of the state dimensions indexed by i.

Cox (doubly-stochastic) process noises would generally take the form

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + N^i_{t+\delta t}(\Lambda^i_{t+\delta t}) - N^i_t(\Lambda^i_t)\,, \tag{1.11}$$

where the rate Λ^i_t is now a sample from some continuous-time stochastic process (in the positive-only domain) for each of the state dimensions indexed by i.

Self-exciting process noises would generally take the form

$$X^i_{t+\delta t} = F^i_{t+\delta t}(X', t) + N^i_{t+\delta t}[\mathcal{I}^i_{t+\delta t}(N', \dots)] - N^i_t[\mathcal{I}^i_t(N'', \dots)]\,, \tag{1.12}$$

where the stochastic rate $\mathcal{I}^i_t(N', \dots)$ now depends on the history of N' explicitly (amongst other potential inputs - see, e.g., Hawkes processes) for each of the state dimensions indexed by i.

Generalised probabilistic discrete state transitions would take the form

$$X^i_{t+\delta t} = /F^i_{t+\delta t}(X', t) + T^i_{t+\delta t}(X')\,, \tag{1.13}$$

where $T^i_{t+\delta t}(X')$ is a generator of the next state to occupy. This generator uses the current state transition probabilities (which are generally conditional on X') at each new step.

1.4 Summary of desirable features

- using the learnings from the previous sections looking at specific example processes
- above formalism is so general that it can do anything - so while it shall serve as a useful guide and reference point, it would be good here to go through more of the specific desirable components we want to have access to in the software itself
- it might not always be convenient to have the windowed histories stored as S but some other varying quantity which can be used to construct S? take fractional brownian motion as an example of this! hence, need to provide more possible input histories into S
- want the timestep to have either exponentially-sampled lengths or fixed lengths in time
- formalism already isn't explicit about the choice of deterministic integrator in time
- but also want to be able to choose the stochastic integrator in continuous processes (Itô or Stratonovich?)

- enable correlated noise terms at the sample generator level
- configurable setup of simulations with just yamls + a single .go file defining the terms

Test cite [1]

1.5 Software design choices

2

Building a Rugby simulator game

Concept. The idea here is

2.1 Introduction

Since the basic game engine will run using the stochadex sampler, the novelties in this project are all in the design of the rugby match model itself. And, in this instance, I'm not especially keen on spending a lot of time doing detailed data analysis to come up with the most realistic values for the parameters that are dreamed up here. Even though this would also be interesting.

One could do this data analysis, for instance, by scraping player-level performance data from one of the excellent websites that collect live commentary data such as rugbypass.com or espn.co.uk/rugby.

This game is primarily a way of testing out the interface of the stochadex for other users to build projects with. This should help to both iron out some of the kinks in the design, as well as prioritise adding some more convenience methods for event-based modelling into its code base.

2.2 Designing the event simulation engine

We need to begin by specifying an appropriate event space to live in when simulating a rugby match. It is important at this level that events are defined in quite broadly applicable terms, as it will define the state space available to our stochastic sampler and hence the simulated game will never be allowed to exist outside of it. So, in order to capture the fully detailed range of events that are possible in a real-world match, we will need to be a little imaginative in how we define certain gameplay elements when we move through the space.

The diagrams below sum up what should hopefully work as a decent initial approximation while providing a little context with specific examples of play action.

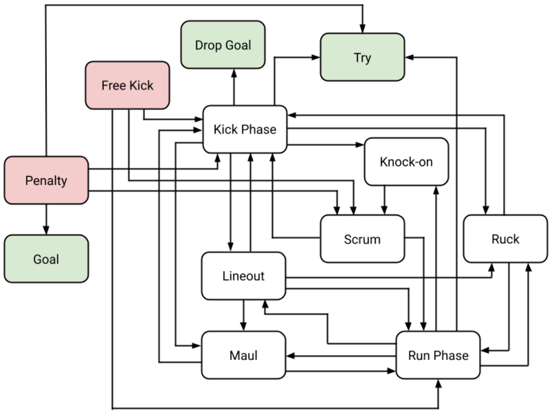
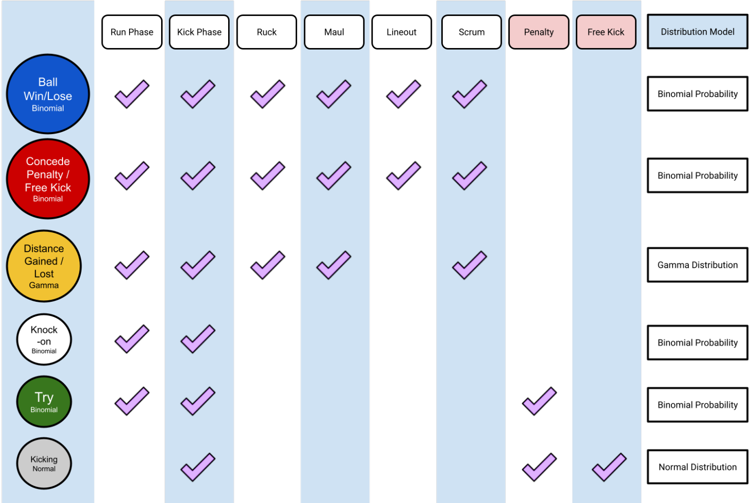


Figure 2.1: Simplified event graph of a rugby union match.

2.3 Linking to player attributes

2.4 Deciding on gameplay actions

2.5 Writing the game itself



	Run Phase	Kick Phase	Ruck	Maul	Lineout	Scrum	Penalty	Free Kick	Distribution Model
Ball Win/Lose Binomial	✓	✓	✓	✓	✓	✓			Binomial Probability
Concede Penalty / Free Kick Binomial	✓	✓	✓	✓	✓	✓			Binomial Probability
Distance Gained / Lost Gamma	✓	✓	✓	✓		✓			Gamma Distribution
Knock -on Binomial	✓	✓							Binomial Probability
Try Binomial	✓	✓					✓		Binomial Probability
Kicking Normal		✓					✓	✓	Normal Distribution

Figure 2.2: Optional model ideas.

Bibliography

Bibliography

[1] E. Dimastrogiovanni, M. Fasiello, R. J. Hardwick, H. Assadullahi, K. Koyama and D. Wands, *Non-Gaussianity from Axion-Gauge Fields Interactions during Inflation*, 1806.05474.